Inside Memory Layout, v4

Based on Marvell SDK for Basalt SoC

Walter Zhou
walterzh@marvell.com

2015-02-24

eROM

The SoC's bootcode is in 0xF70X,XXXX

Load ELF from USB printer channel to the high-end of physical memory.

pBufferMem = DRAMLoadAddr(be32_to_cpu(size));

(g_PlatformMemSize – (1000000 + **X** + 0x100)) & 0xffffff00

X is elf size

Bootcode ROM info

$ readelf –l BOOTCODE_ROM.elf

Elf file type is EXEC (Executable file)

Entry point 0xf7000000

There are 4 program headers, starting at offset 52

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
LOAD	**0x008000**	**0xf7000000**	**0xf7000000**	**0x0a904**	**0x0a904**	**R E**	**0x8000**
LOAD	0x013370	0xf7203370	0xf700a908	0x003b0	0x003b0	RW	0x8000
LOAD	0x018100	0xf7200100	0xf7200100	0x02dc0	0x0326c	RW	0x8000
LOAD	0x020000	0xff200000	0xff200000	0x0006c	0x0006c	RW	0x8000

Section to Segment mapping:

Segment Sections...

00 .vectors .text .ram.text .rodata .rodata.str1.4 .glue_7 .mpu.text

01 .data

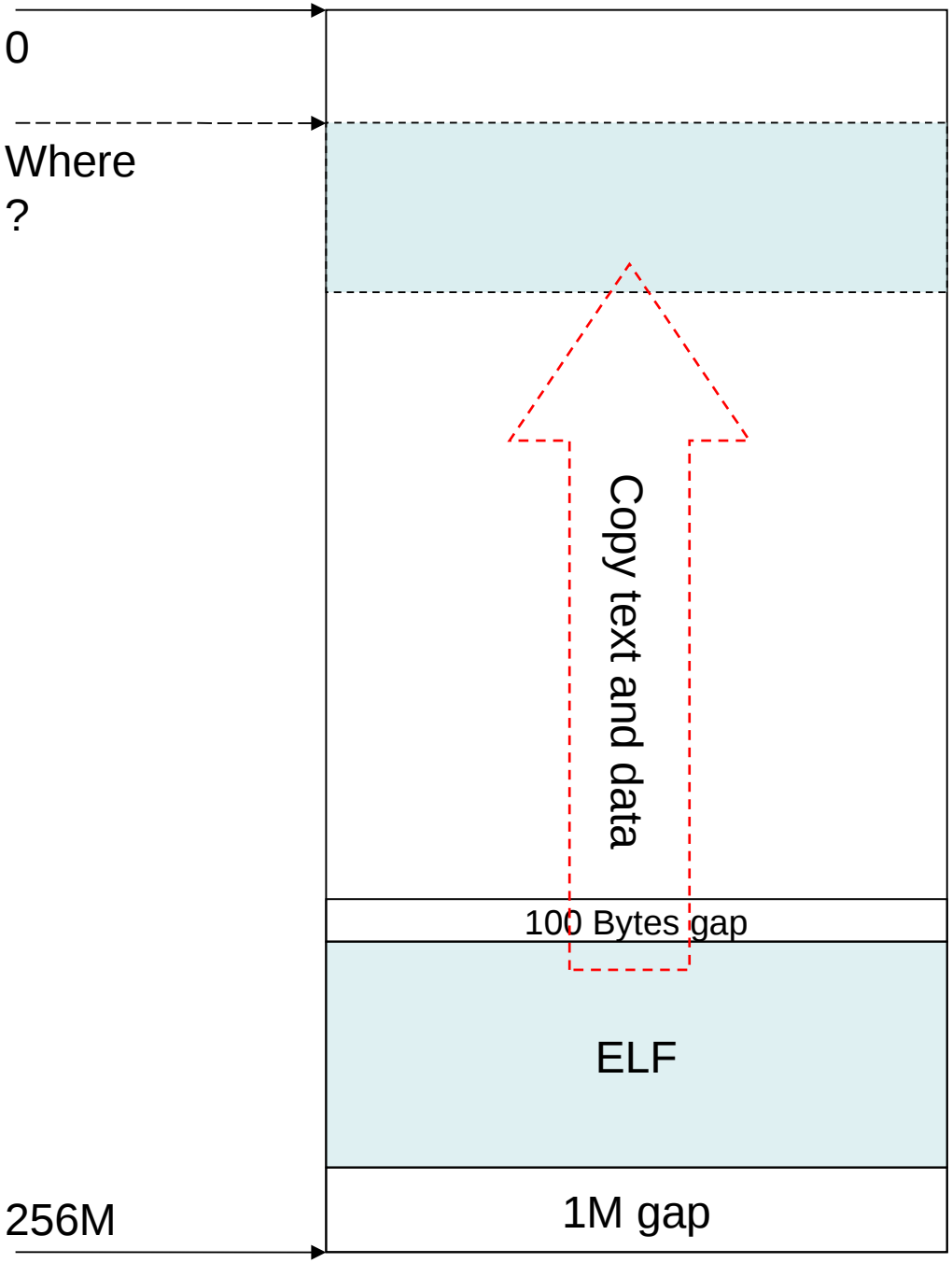
02 .lcm .bss

03 .lcmNonCache

Bootcode ROM info

```
MEMORY
{
 /* ram (wx) : ORIGIN = 0x000000, LENGTH = 128K */

/* main processor ROM */
  rom(rx) : ORIGIN =  0xF7000000, LENGTH = 64k
/* M3 processor RAM */
  tcm (wx) : ORIGIN = 0xF7100000 , LENGTH = 8K
/* main processor RAM */
/* put the TABLE of key VARs into noncache space.  Allows m3 to get access to the
     address */
  lcmNonCache (wx) : ORIGIN = 0xFF200000 , LENGTH = 256
  lcm (wx)         : ORIGIN = 0xF7200100 , LENGTH = 128K


}
```

SS_MemMapROM.ld



LoadELF() copy the text and data in the ELF file to the specific address according to “Program Header”

CodeDstAddr = secHeader.sh_addr;

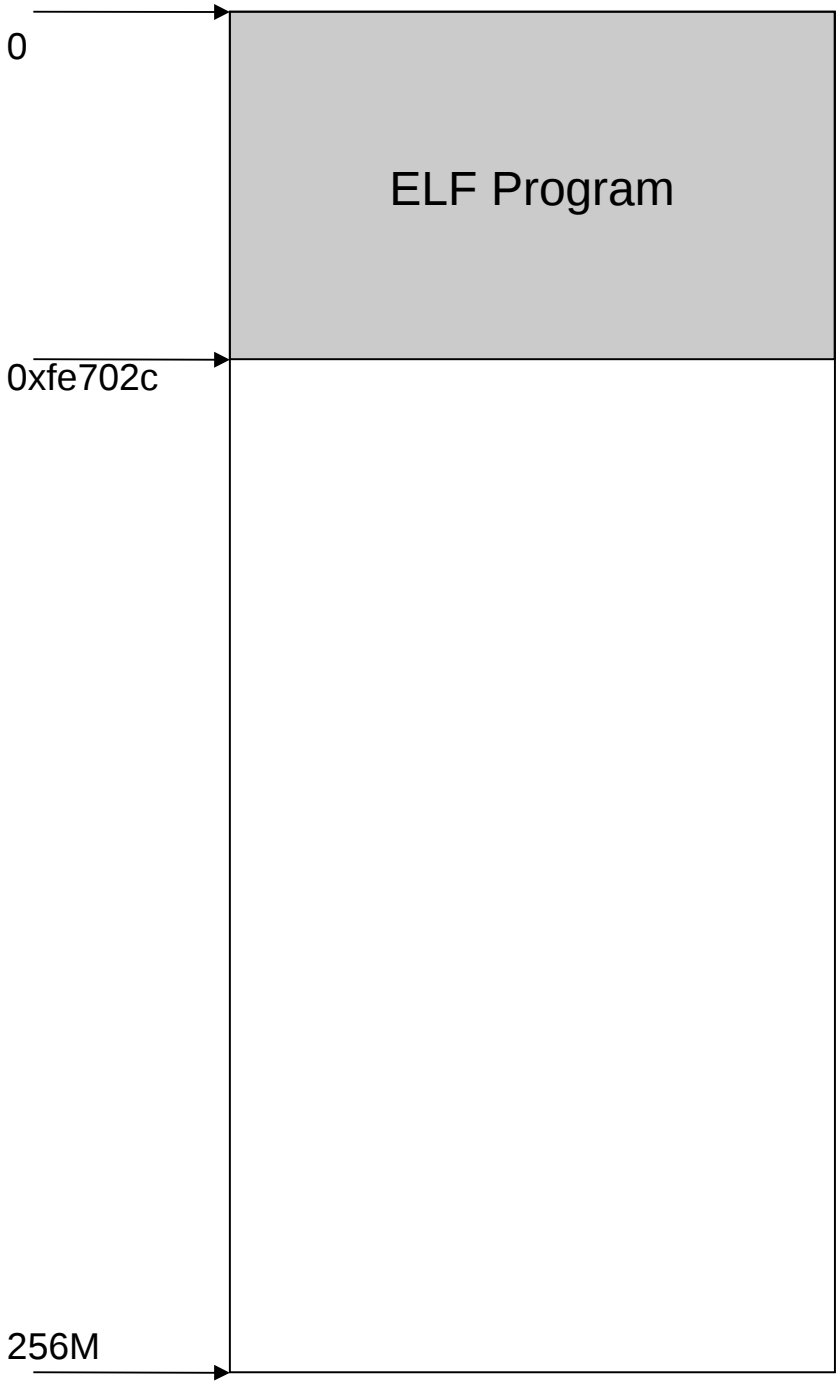
codeSrcAddr = imageStartAddr + secHeader.sh_offset;

MEMCPY(codeDstAddr, codeSrcAddr, secHeader,sh_size);

ELF info

$ readelf –l marvell_6110_mfp_sdk-debug_stripped.elf

```
Elf file type is EXEC (Executable file)
Entry point 0x0
There are 3 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz  MemSiz   Flg Align
  EXIDX          0xb94578 0x00b94478 0x00b94478 0x3cd88  0x3cd88  R   0x4
  LOAD           0x000100 0x00000000 0x00000000 0xc82690 0xfe702c RWE 0x100
  LOAD           0xc827a0 0x00fe7040 0x00fe7040 0x21d212 0x21d212 R   0x20

 Section to Segment mapping:
  Segment Sections...
   00     .ARM.exidx
   01     vectors text .test .ARM.extab .ARM.exidx data bss icache_align
   02     rodata
```



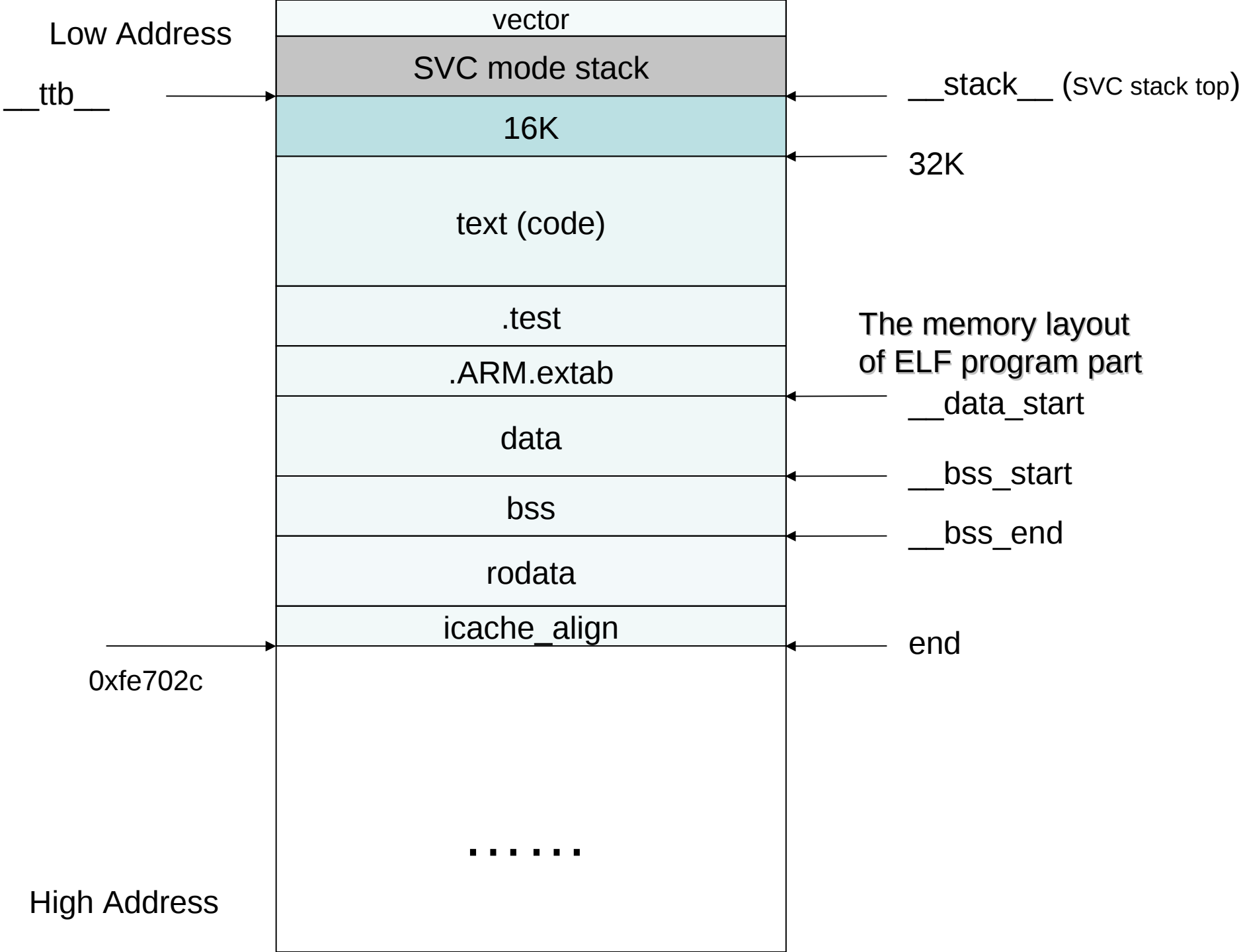
The ELF loader in eROM only copy the Program part from ELF image to address 0.

But how to arrange memory layout in the “ELF Program” ?

CodeDstAddr = 0;
codeSrcAddr = imageStartAddr + secHeader.sh_offset;

MEMCPY(**0**, codeSrcAddr,**0xfe702c**);

```
$ readelf –S marvell_6110_mfp_sdk-debug_stripped.elf
There are 12 section headers, starting at offset 0xe9fa3c:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] vectors           PROGBITS        00000000 000100 000020 00  AX  0   0  4
  [ 2] text              PROGBITS        00008000 008100 b31a70 00 WAX  0   0 256
  [ 3] .test             PROGBITS        00b39a70 b39b70 0000fc 00  WA  0   0  4
  [ 4] .ARM.extab        PROGBITS        00b39b9c b39c9c 05a8dc 00   A  0   0  4
  [ 5] .ARM.exidx        ARM_EXIDX       00b94478 b94578 03cd88 00  AL  2   0  4
  [ 6] data              PROGBITS        00bd1200 bd1300 0b1490 00  WA  0   0 32
  [ 7] bss               NOBITS          00c826a0 c82790 36498c 00  WA  0   0 32
  [ 8] rodata            PROGBITS        00fe7040 c827a0 21d212 00   A  0   0 32
  [ 9] .ARM.attributes   ARM_ATTRIBUTES  00000000 e9f9b2 00002d 00      0   0  1
  [10] icache_align      PROGBITS        00b39b80 b39c80 00001c 00  AX  0   0  4
  [11] .shstrtab         STRTAB          00000000 e9f9df 00005c 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```


Low Address
vector
SVC mode stack
__ttb__
__stack__ (SVC stack top)
16K
32K
text (code)
.test
The memory layout
of ELF program part
.ARM.extab
__data_start
data
__bss_start
bss
__bss_end
rodata
icache_align
end
0xfe702c
.....
High Address

```
MEMORY
{
    ram (rwx) : ORIGIN = 0x0, LENGTH = 64M
    spi (rx)  : ORIGIN = 0xfe000000, LENGTH = 16M
}

OUTPUT_ARCH(arm)
ENTRY(reset)
EXTERN(reset)         /* make sure this gets linked in */

SECTIONS
{
    spibase :
    {
        BSPI_BASE = .;
    } > spi
    /* Place vector table at offset 0x0 */
    vectors 0x0 :
    {
        *(.text.vectors)
    } > ram
…….
```

Please refer to buildtools/memory_map.ld linker script.

```
for(int i = 0; i < 10; i++)
{
    a[i] = I;
    ……
}
```

→ text (code)

```
int a = 9;
static int b = 0;
```

→ data

```
int a;
static int b;
```

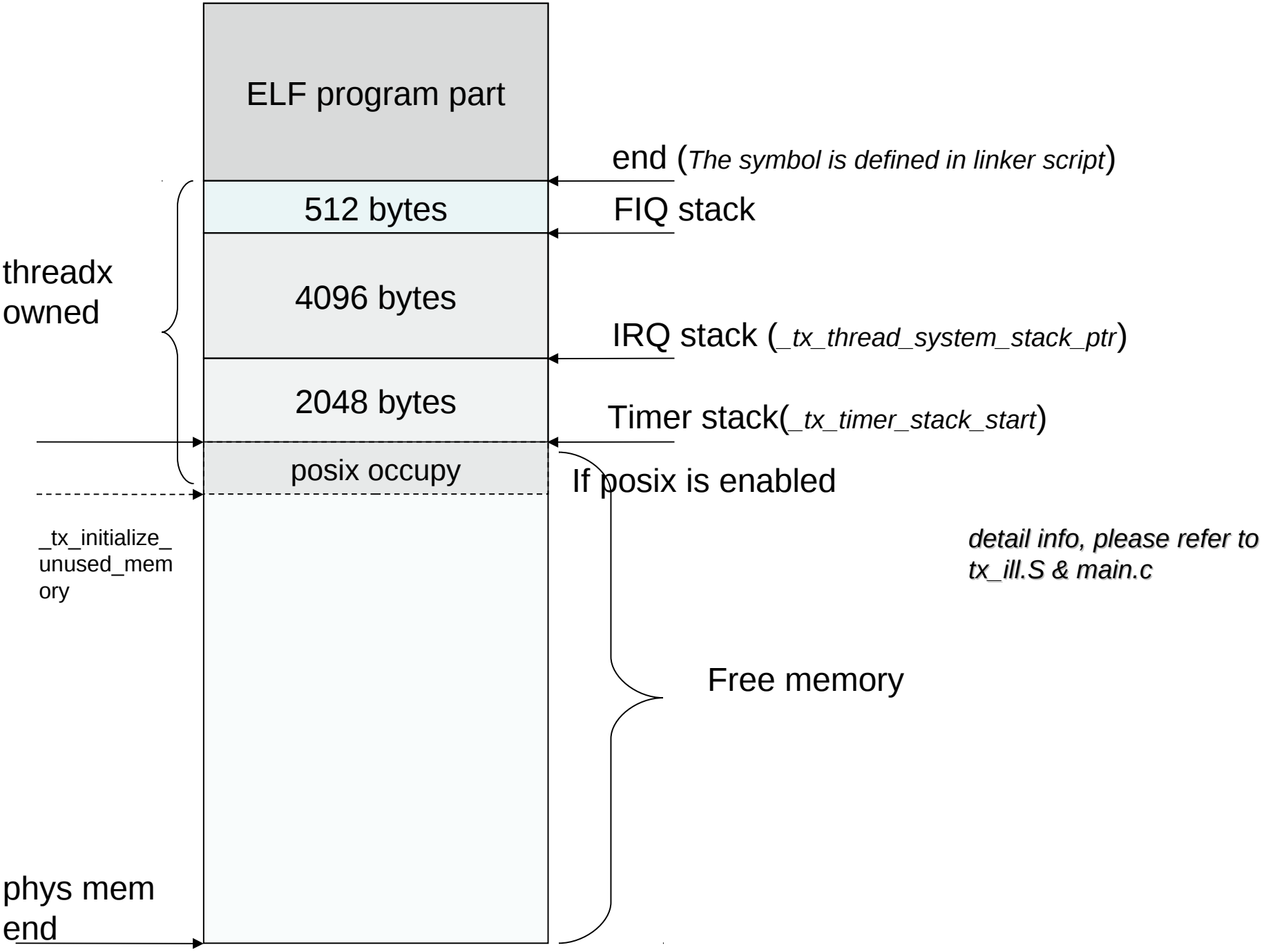
→ bss

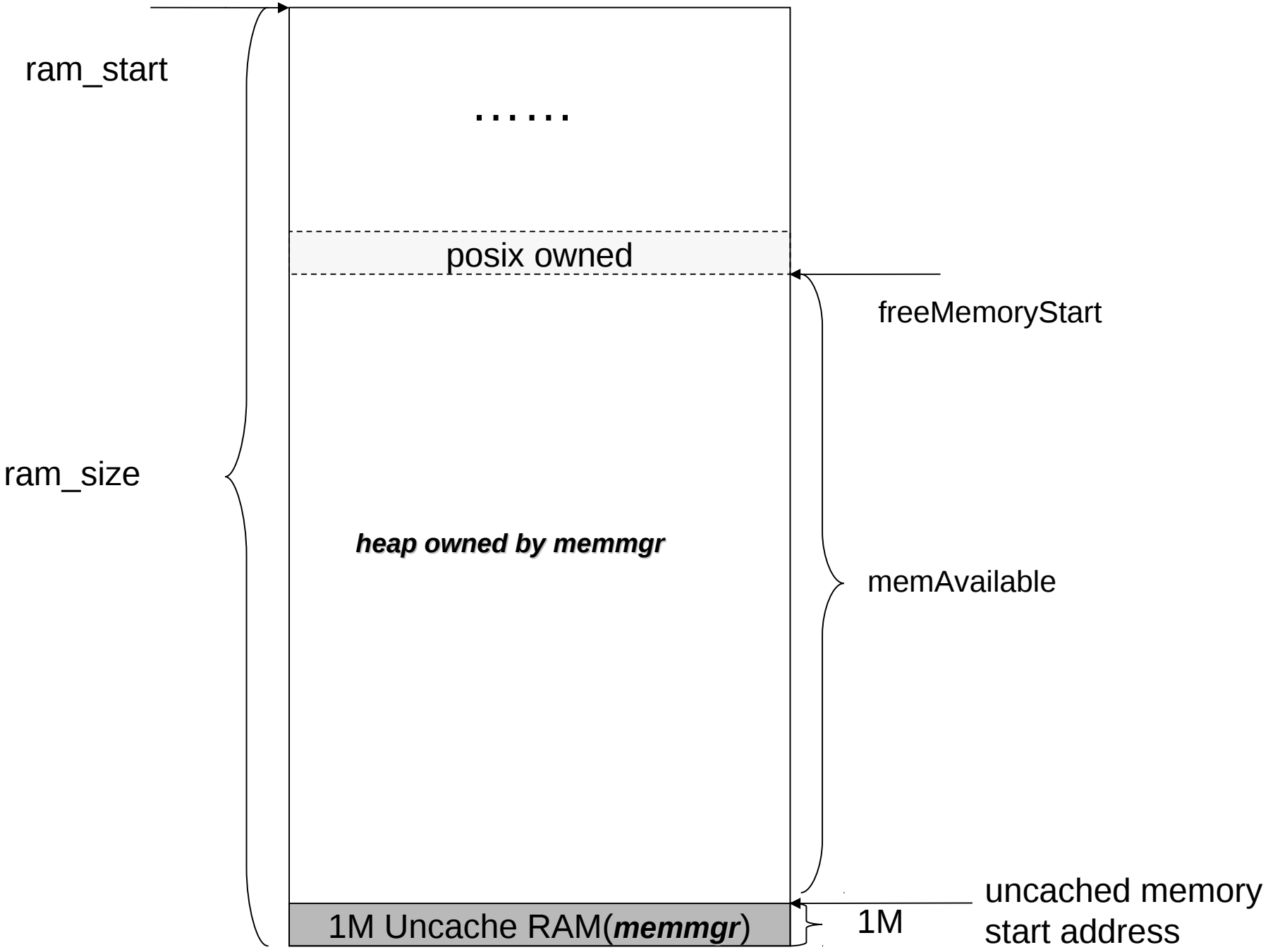
```
const int a = 9;
static const int b = 0;
```

→ rodata

```
UNIT_TEST("cdma", cdma_test);
UNIT_TEST("engine", engine_test);
```

→ .test


ELF program part
end (The symbol is defined in linker script)
512 bytes
FIQ stack
threadx
owned
4096 bytes
IRQ stack (_tx_thread_system_stack_ptr)
2048 bytes
Timer stack(_tx_timer_stack_start)
posix occupy
If posix is enabled
_tx_initialize_
unused_mem
ory
detail info, please refer to
tx_ill.S & main.c
Free memory
phys mem
end


ram_start
.....
posix owned
freeMemoryStart
ram_size
heap owned by memmgr
memAvailable
uncached memory
start address
1M Uncache RAM(memmgr)
1M

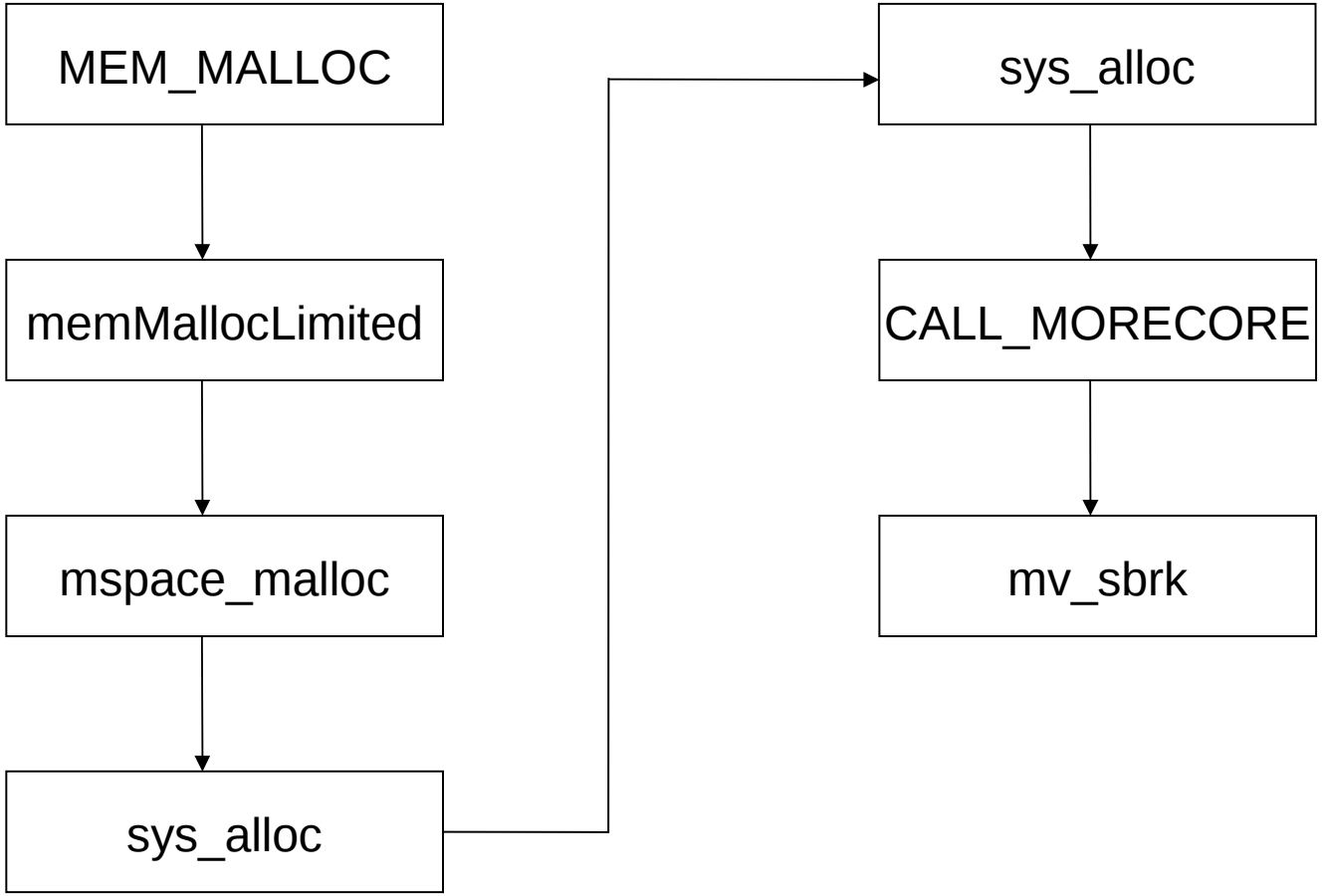
memmgr init

1. memInitMemory(freeMemoryStart, memAvailable);
2. memInitUncached(hwGetUncachedRamStartAddress() , hwGetUncacheRamSize())

hwGetUncachedRamStartAddress() is 255M

hwGetUncacheRamSize() is 1M

Interfaces between dlmalloc and threadx



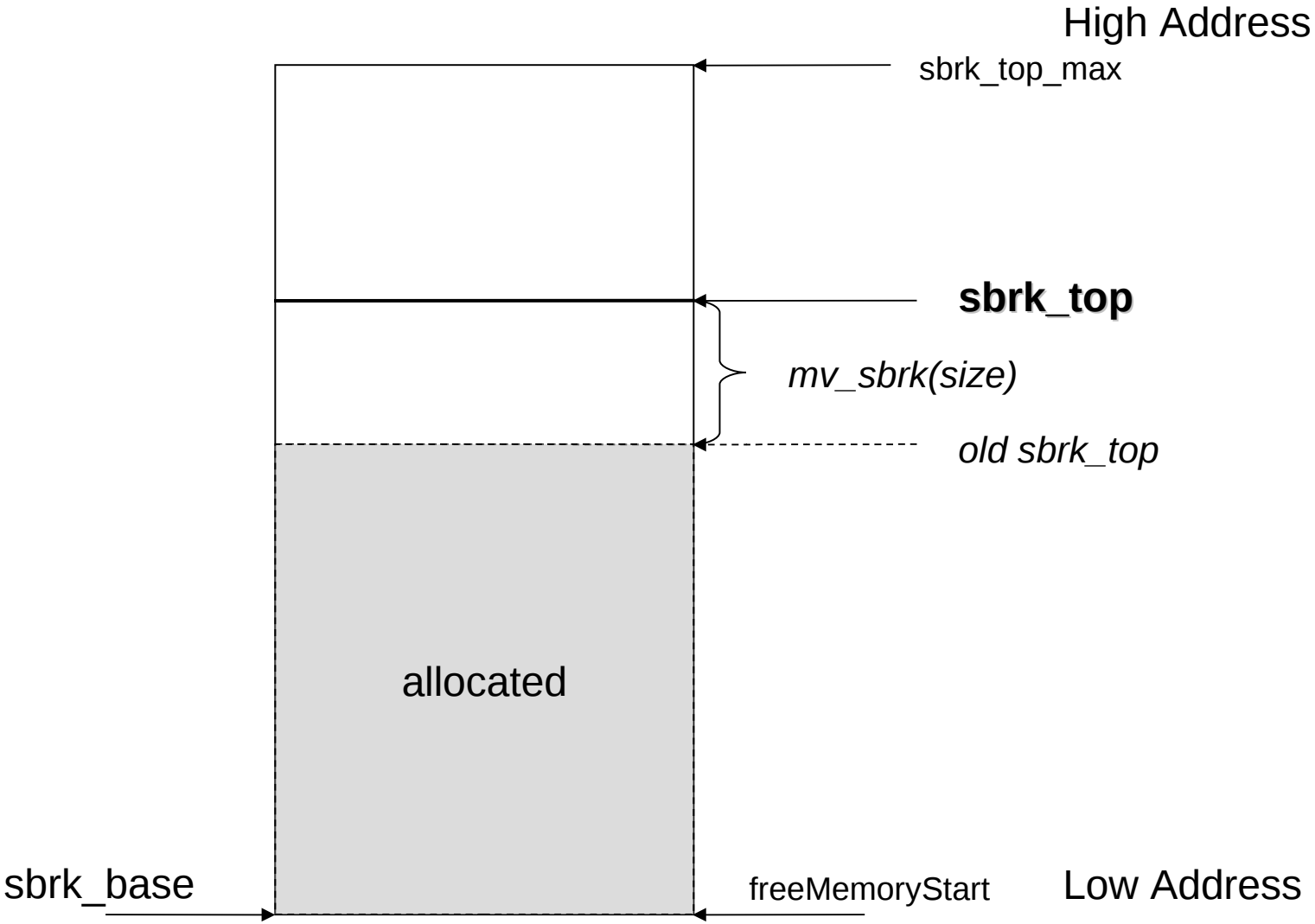
Interfaces between dlmalloc and threadx

```
void *mv_sbrk(int size)
{
   void *ptr = 0;

   if (sbrk_top == 0)
      sbrk_top = sbrk_base;

   if (size > 0)
   {
      if ((uintptr_t)sbrk_top + size <= (uintptr_t)sbrk_top_max)
      {
         ptr = sbrk_top;
//#ifdef DEBUG
         memset(ptr, 0xCD, size);  // set the memory to a known pattern DAB
//#endif
         sbrk_top = (char*)ptr + size;
         return ptr;
      }
      return (void *) MFAIL;
   }
   else if (size < 0)
   {
      return (void *) MFAIL;
   }
   else
   {
      return sbrk_top;
   }

   return 0; // compiler warning bug! DAB
}
```

Interfaces between dlmalloc and threadx

```
// initialize the heap starting at base for size bytes.
void mv_sbrk_init(void *base, size_t size)
{
    sbrk_top = 0; // Necessary to allow subheap creation to succeed! DAB
    sbrk_base = base;
    sbrk_top_max = (char *) base + size;
}
```

Interfaces between dlmalloc and threadx



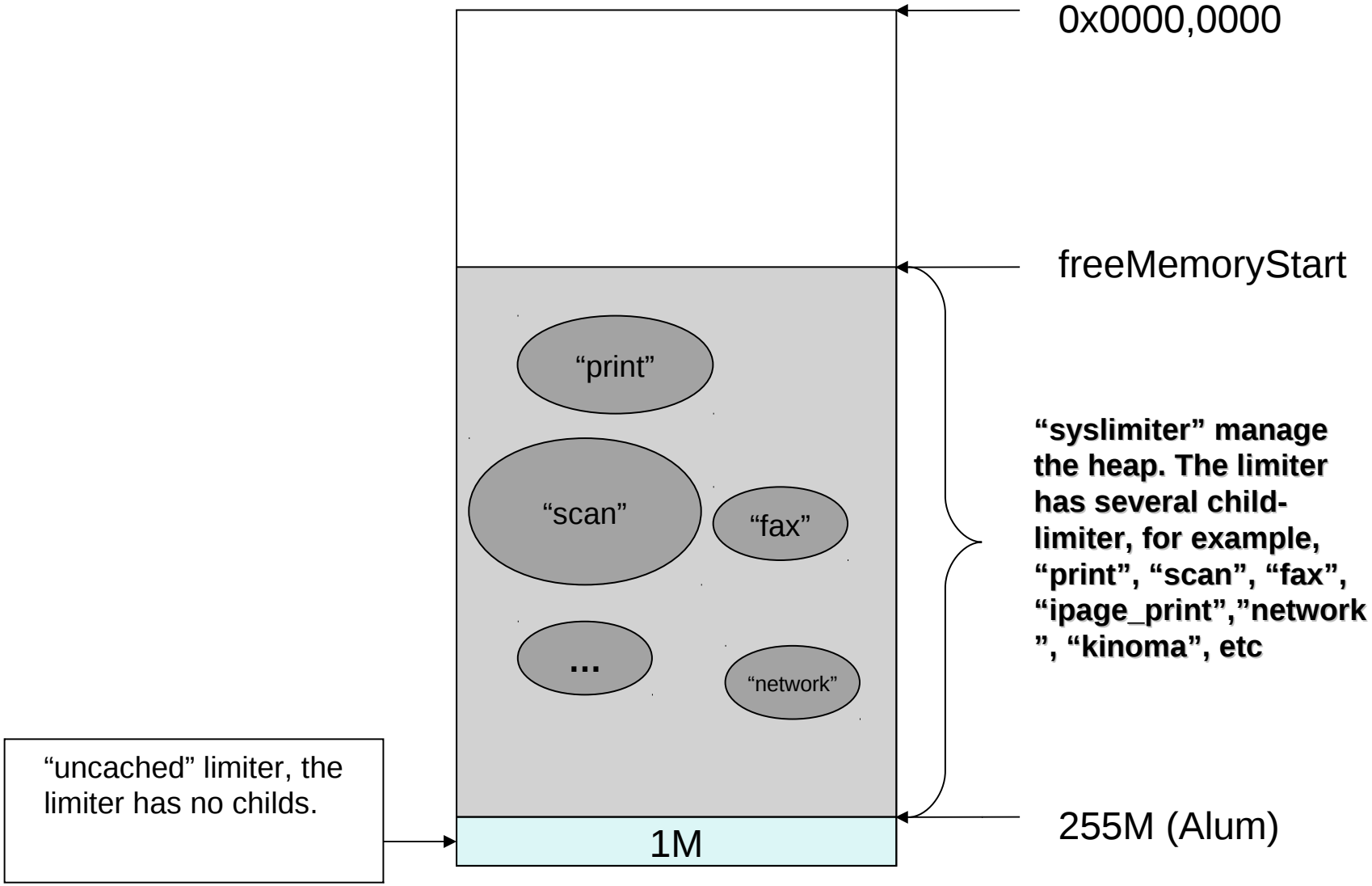
Interfaces between dlmalloc and threadx

memmgr needs synchronization mechanisms provided by threadx OS

```
#include <tx_api.h>
#define MLOCK_T   TX_MUTEX
#define INITIAL_LOCK(l)     tx_mutex_create(l, “dlmalloc mutex”, TX_NO_INHERIT)
#define ACQUIRE_LOCK(l) tx_mutex_get(l, TX_WAIT_FOREVER)
#define RELEASE_LOCK(l) tx_mutex_put(l)
```

Notes: If you want to port dlmalloc to other OS, you need define the upper macros!

Memory Limiter



mlimiter_t

```
struct mlimiter_s
{
   void *ms;            /**< This value must be first. ms represents the
                        memory manager "mspace," or rather, the heap.
                        In most cases this will be the base system heap. */

   uint32_t max;        /**< The maximum number of bytes the user of this
                        limiter can allocate.  Allocations used in
                        conjunction with this limiter will fail once
                        the allocations made with this limiter reach
                        this limit. This is the active maximum.
                        Note it is possible that the max is lower than the
        current,
                        this happens when a pool is shrunk ie has its limit
        lowered.
                        */

   uint32_t highest_max;   /**< max will always be <= highest max,
                        the maximum amount of memory needed. */

   uint32_t lowest_max;    /**< max will always be >= lowest max,
                        the miniumum amount of memory needed.
                        A parent limiter will reserve lowest_max,
                        this will never be availiable to its children.
                        */

   mlimiter_low_memory_strategy_t strategy; /**< null or a callback on
        memory

   uint32_t active;        /**< active limiter
                        set by mlimiter_start() mlimiter_stop()
                        inactive limiters can still have memory but
                        allocations out of an inactive limiter will
                        have a high probability of being out of memory.
                        */

   uint32_t current;       /**< Current in use bytes (bytes allocated in
                        conjunction with this limiter).
                        read only!
                        */

   uint32_t high_water;    /**< The limiter's high water mark, in bytes.
                        This value represents the highest number of bytes
                        allocated at one time in conjunction with this
                        limiter.
                        read only!
                        */

   mlimiter_t *parent;     /**< This value represents the limiter's parent.
                        if NULL this is a base/parent heap.
                        one level of childred only,
                        set at startup time read only afterwards.
                        */

};
```

mlimiter_config.c

```
/// syslimiter always at position 1
  {
      .ms = 0,
      .max = 0xffffffff,        //
      .highest_max = 0xffffffff, // as much as possible.
      .lowest_max = 0x200000,    // 2 meg minimum must be left over for system heap, Reserve.
      .strategy = 0,  // must stay 0, assert based
      .active = 0,
      .current = 0,
      .high_water = 0,
      .parent = 0,
  },
/// print
  {
      .ms = 0,                    // set at runtime
      .max = 0,                   // set at runtime
      .highest_max = 0xffffffff,    // as much as possible
      .lowest_max = 0x100000,       // minimum ram
      .strategy = print_low_memory_strategy,  // may have a low memory strategy
      .active = 0,                // set at runtime
      .current = 0,               // set at runtime
      .high_water = 0,            // set at runtime
      .parent = 0,                // set at runtime, will be syslimiter
  },
```

“syslimiter” init

memInitMemory()

↓

memCreateSysHeap()

↓

```
sysHeap = create_mspace(size, 1);
   ASSERT(sysHeap != 0);
   sysLimiter = mlimiter_by_name("syslimiter");
   ASSERT(sysLimiter);
   sysLimiter->ms = sysHeap;
   sysLimiter->max = size;
   sysLimiter->highest_max = size;
   sysLimiter->strategy = 0;
   sysLimiter->active = 1;
   sysLimiter->current = 0;
   sysLimiter->high_water = 0;
   sysLimiter->parent = 0;
```

size is memAvailable

sub-limiter init

```
ASSERT(size > sysLimiter->lowest_max);
  size -= sysLimiter->lowest_max;

  // initialize all the limiters.

  int i=2;
  mlimiter_t *lim = mlimiter_by_index(i);
  for (;lim ;  lim = mlimiter_by_index(++i))
  {
      lim->ms = sysHeap;
      lim->max = size > lim->max ? lim->max : size;
      lim->parent = sysLimiter;
  }
```

dump limiter

```
CMD==> memory limiter
LIM [uncached] (00c884e4)
  :      .ms 0x0ff00008     max 01039360
  :     h_max   01039360   l_max 01039360  strategy 0x00000000
  :    active   00000000  parent 0x00000000
  : highwater   00045744 current 00045696
LIM [syslimiter] (00c88508)
  :      .ms 0x013fc1e0     max 246422056
  :     h_max   246422056   l_max 02097152  strategy 0x00000000
  :    active   00000001  parent 0x00000000
  : highwater   02099128 current 02051192
LIM [sys_retry_forever] (00c8852c)
  :      .ms 0x013fc1e0     max 02097152
  :     h_max   02097152   l_max 00065536  strategy 0x00663e44
  :    active   00000001  parent 0x00c88508
  : highwater   00002064 current 00000000
LIM [print] (00c88550)
  :      .ms 0x013fc1e0     max 00000000
  :     h_max   -0000001   l_max 01048576  strategy 0x003920e4
  :    active   00000000  parent 0x00c88508
  : highwater   00000000 current 00000000
LIM [ipage_print] (00c88574)
  :      .ms 0x013fc1e0     max 244324904
  :     h_max   -0000001   l_max 01048576  strategy 0x003920e4
  :    active   00000000  parent 0x00c88508
  : highwater   00000000 current 00000000
LIM [scan] (00c88598)
  :      .ms 0x013fc1e0     max 241818152
  :     h_max   -0000001   l_max 01048576  strategy 0x00000000
  :    active   00000000  parent 0x00c88508
  : highwater   00000000 current 00000000
LIM [fax] (00c885bc)
  :      .ms 0x013fc1e0     max 00000000
  :     h_max   08388608   l_max 00000000  strategy 0x00663c70
  :    active   00000000  parent 0x00c88508
  : highwater   00000000 current 00000000
LIM [thumbnails] (00c885e0)
  :      .ms 0x013fc1e0     max 00000000
  :     h_max   08388608   l_max 01048576  strategy 0x00663c70
  :    active   00000000  parent 0x00c88508
  : highwater   00000000 current 00000000
LIM [network] (00c88604)
  :      .ms 0x013fc1e0     max 00409600
  :     h_max   00409600   l_max 00262144  strategy 0x00663c70
  :    active   00000001  parent 0x00c88508
  : highwater   00025552 current 00025552
LIM [kinoma] (00c88628)
  :      .ms 0x013fc1e0     max 00000000
  :     h_max   16777216   l_max 02097152  strategy 0x00663c70
  :    active   00000000  parent 0x00c88508
  : highwater   00000000 current 00000000
```

memory allocation

```
void* mspace_malloc(mlimiter_t *lim, size_t bytes) {
  assert(lim);
  mstate ms = (mstate)lim->ms;

  ......

  if (!PREACTION(ms)) {
    void* mem = 0;
    size_t nb = (bytes < MIN_REQUEST)? MIN_CHUNK_SIZE :
      pad_request(bytes);

    if (lim->max < lim->current + nb) {
        goto postaction;
    }

 ......
```

When relevel memory ?

1. **mlimiter_start()**
2. **Mlimiter_stop()**
3. **printer_low_memory_strategy()**

```
void mlimiter_relevel_memory( mlimiter_t *limiter )
{
   uint32_t current;   // Sum of all active limiters MAX(minimums, current)
   uint32_t min;       // Sum of all active limiters minimums
   uint32_t max;       // Sum of all active limiters maximums
   mlimiter_t *sys_lim;

         ......

   mlimiter_job_current(&max, &current, &min);

         ......

   else
   {
      // Memory exists beyond minimums - and current allocations.
      // Decide how to share this memory
      mlimiter_assign_available_memory( sys_lim->max - current, true );
   }
```

“relevel” algorithm

pending_factories is how many active sub-limiters?

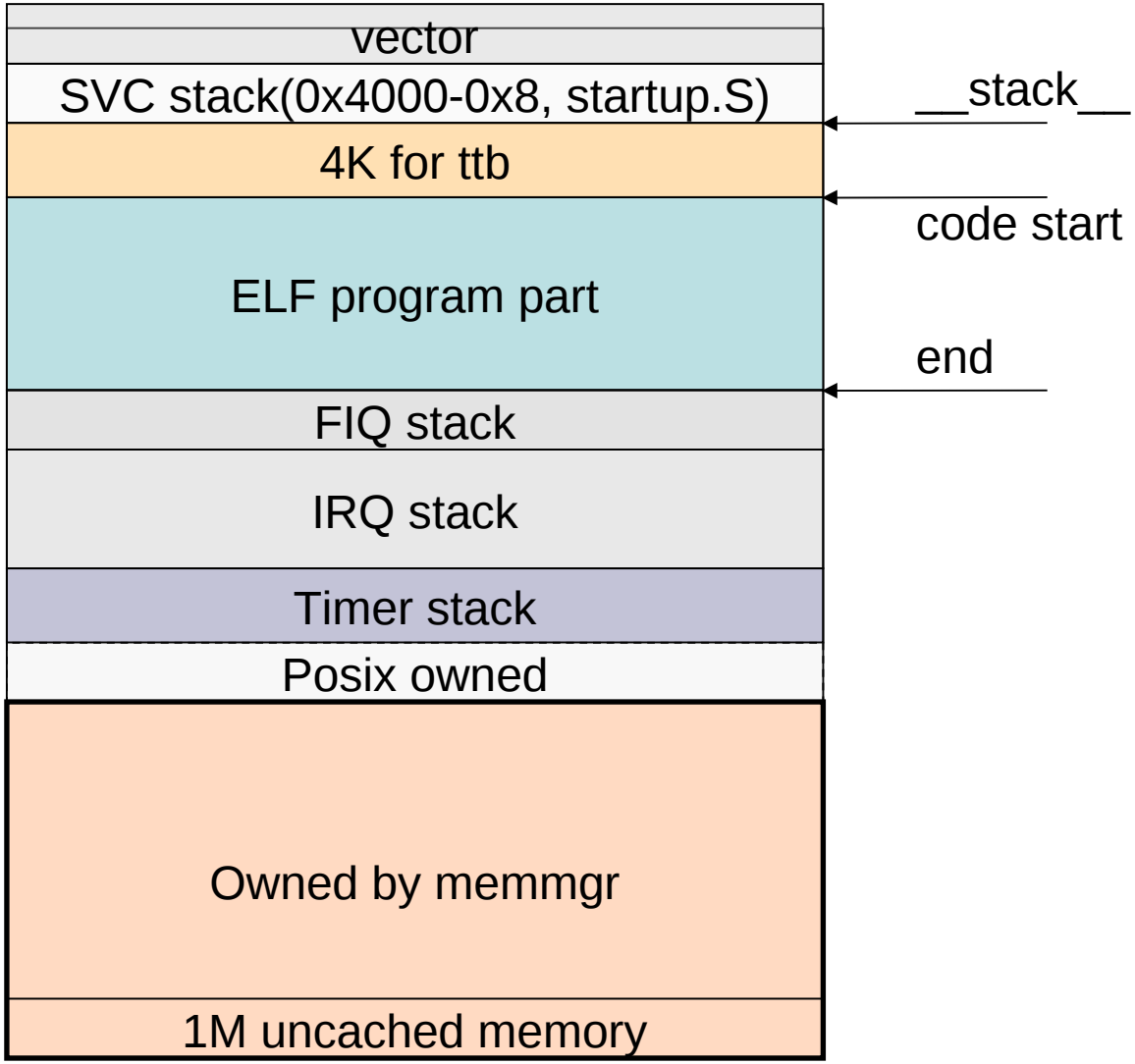
min_spread is the minimum allocable memory of the specific sub-miniter

min_assignment is the allocated memory

min_spread = lim->highest_max - min_assignment;

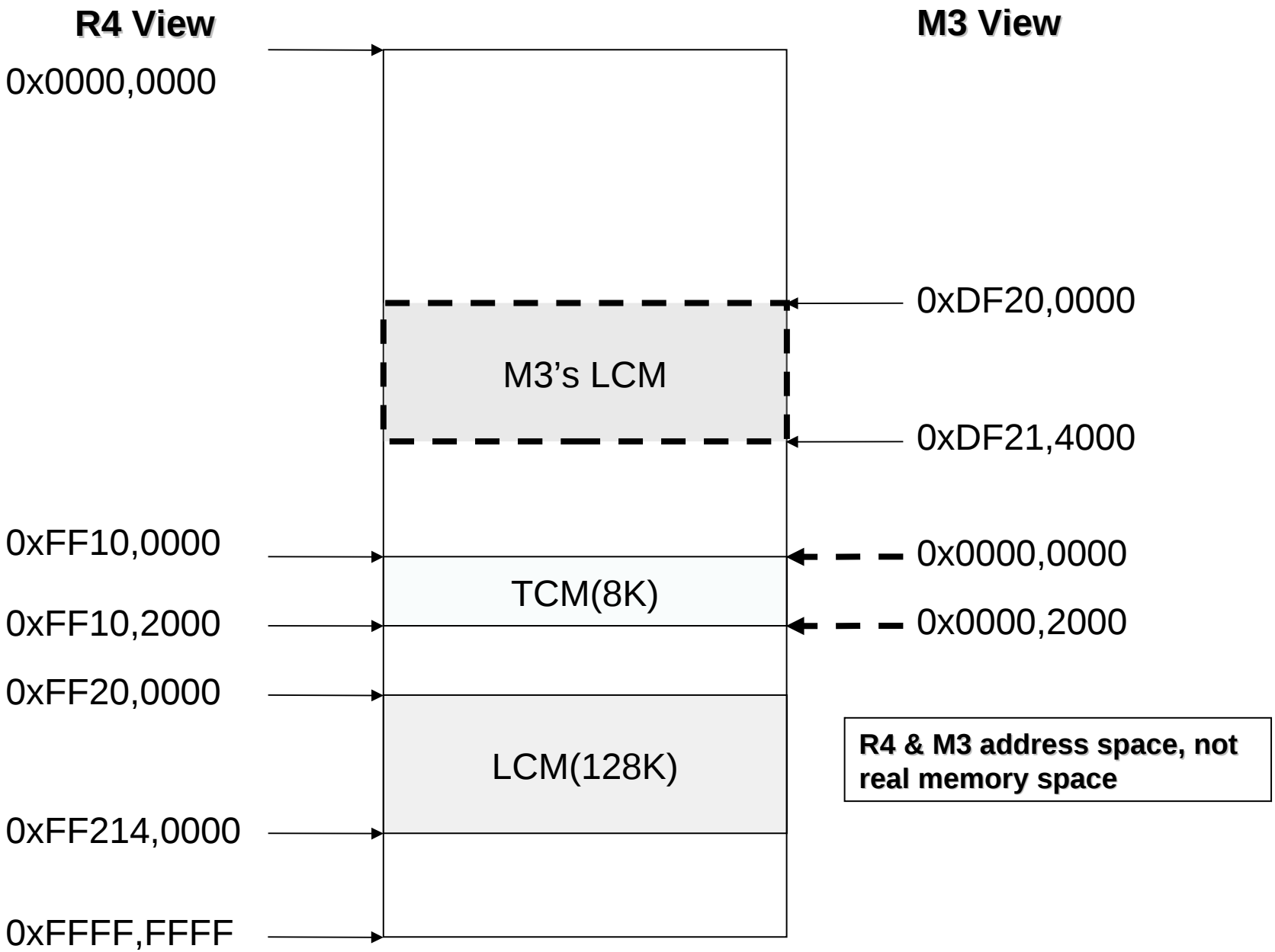
```
fair_share = pool / pending_factories;

    ……

      if ( fair_share > min_spread )
      {
          // There is more available for this factory than it needs
          next_lim->max = next_lim->highest_max;
          pool -= next_lim->highest_max - min_assignment;
      }
      else
      {
          // This factory wants more than its equal share
          next_lim->max = min_assignment + fair_share;
          pool -= fair_share;
          all_at_max = false;
      }
```

overview


vector
SVC stack(0x4000-0x8, startup.S)
4K for ttb
ELF program part
FIQ stack
IRQ stack
Timer stack
Posix owned
Owned by memmgr
1M uncached memory
__stack__
code start
end

Memory Layout in Low Power Mode

R4 & M3 view


R4 View
M3 View
0x0000,0000
0xDF20,0000
M3's LCM
0xDF21,4000
0xFF10,0000
0x0000,0000
TCM(8K)
0xFF10,2000
0x0000,2000
0xFF20,0000
LCM(128K)
R4 & M3 address space, not real memory space
0xFF214,0000
0xFFFF,FFFF

M3 executables

There are 2 executables for LPP (M3).

1. lpp_ucode_boot_bin
2. lpp_ucode_bin

lpp_ucode_boot_bin is created by lpp_boot.S

lpp_ucode_bin is created by
marvell_6110_lowpower_release.elf

lpp_ucode_boot_bin

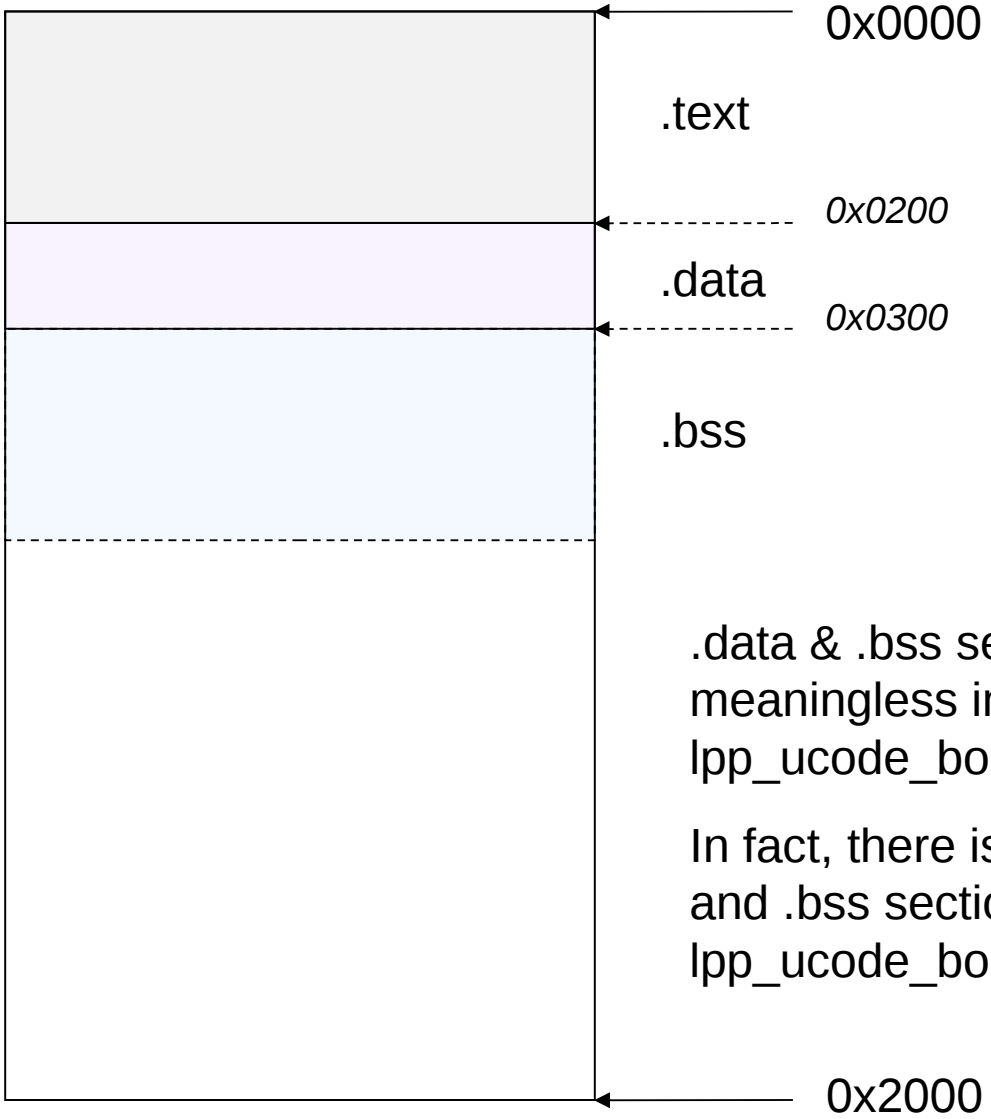
The code in TCM (lpp_boot.S) is very simple,

It only set MPU registers of M3 and then jump to the code in LCM (lpp_ucode_bin)

M3 TCM layout (lpp_ucode_boot.bin)

0xFF10,0000

(R4)

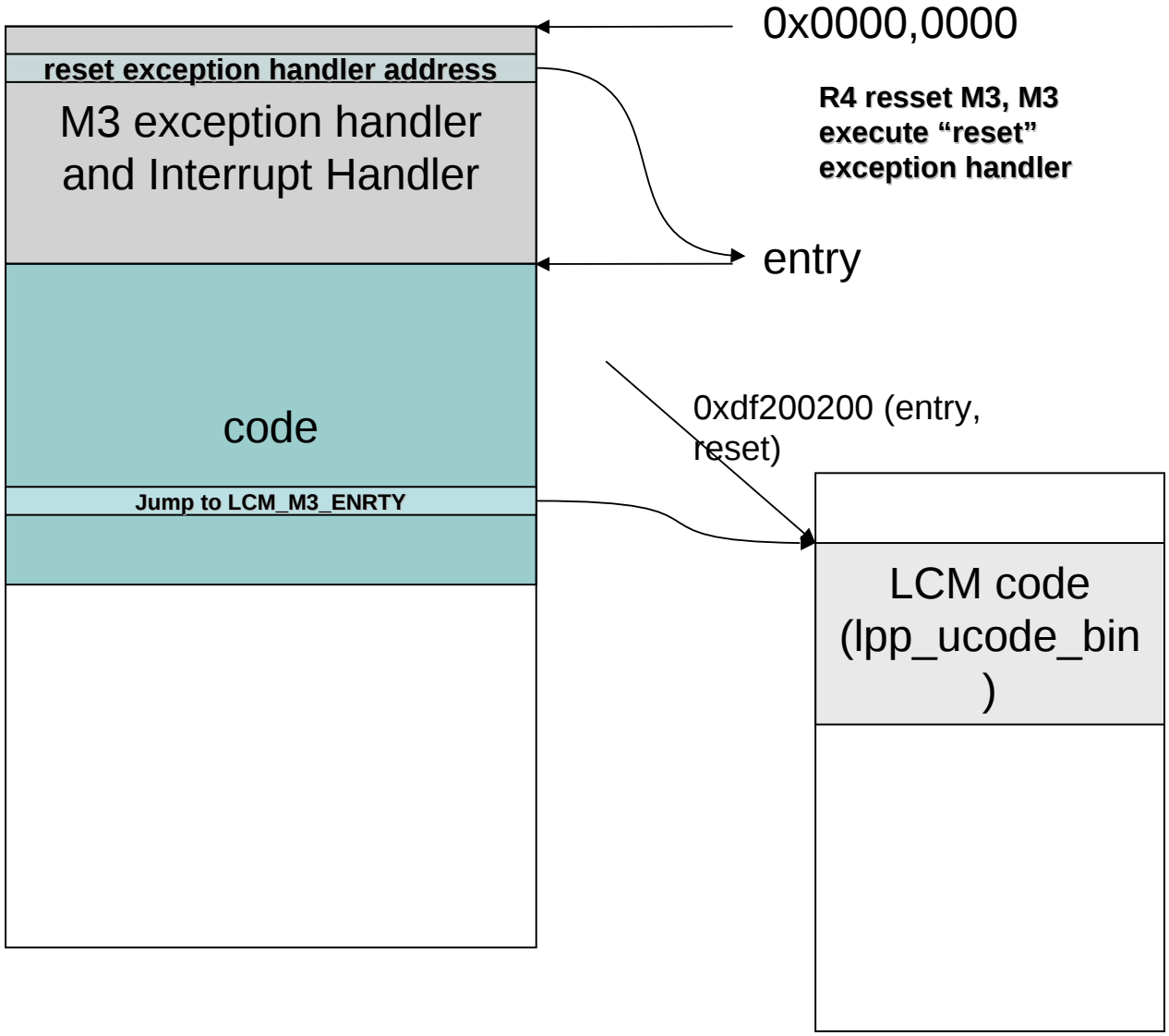


TCM size is 8K

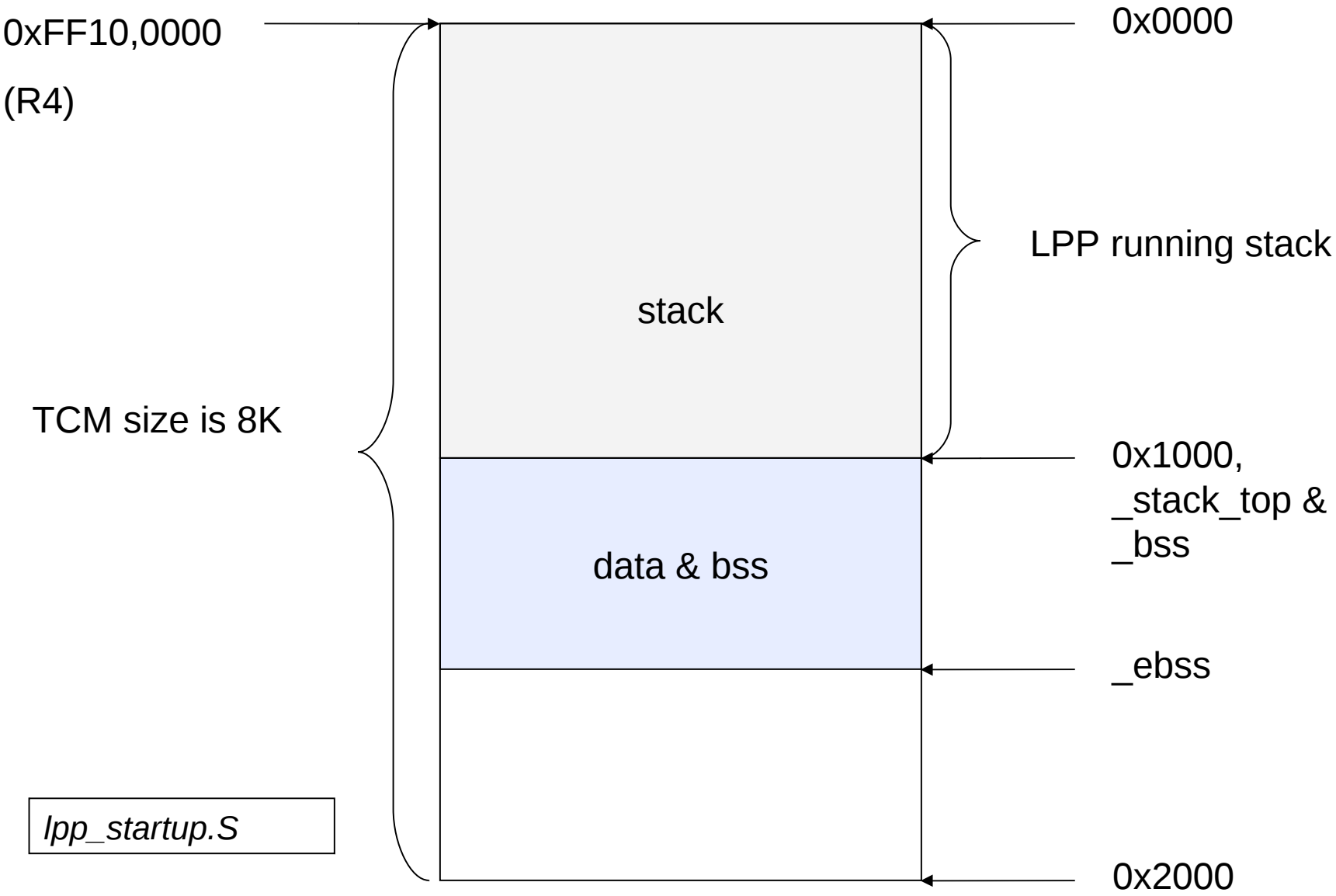
.data & .bss sections are meaningless in lpp_ucode_boot.bin

In fact, there is no .data and .bss section in lpp_ucode_boot.bin

M3 TCM code layout



M3 TCM layout (lpp_ucode.bin)

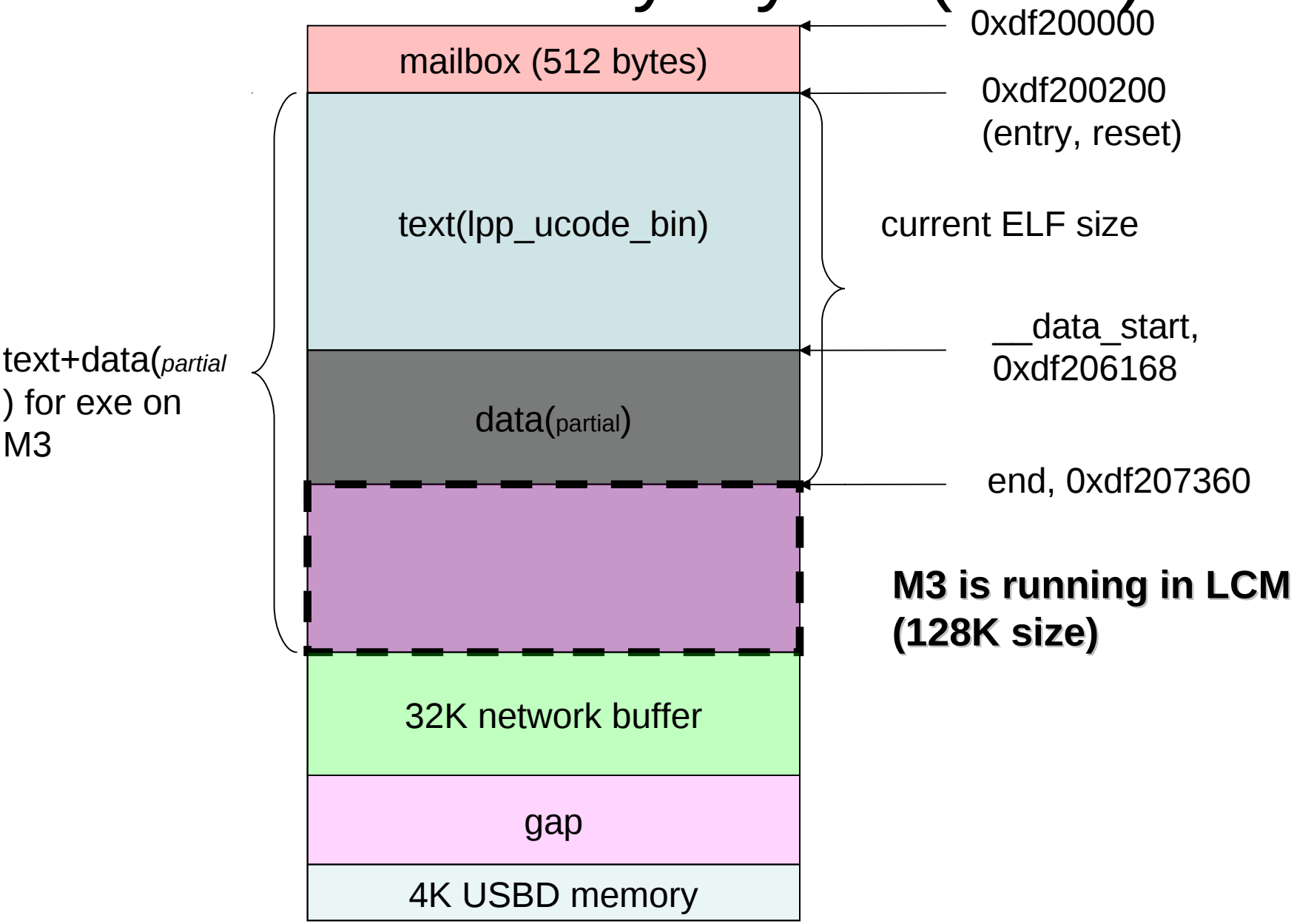


data & bss

$ nm marvell_6110__lowpower-release.elf | sort

```
00001000 B _bss
00001000 B __bss_start__
00001000 b hw_config_table
00001000 B _stack_top
00001014 b blink.5543
00001018 b cp_blink
0000101c b cp_active
00001020 b cp_ack
00001024 b wake_mode
00001028 b lpp_main_flags
0000102c b task_immediate
00001030 b task_list
……
```

M3 memory layout (LCM)



How R4 transfers to M3

1. R4 load lpp_ucode_boot_bin to TCM (R4 view, 0xFF10,0000)
2. R4 load lpp_ucode_bin to LCM(R4 view, 0xFF20,0000)
3. R4 save all registers (in all kinds of modes) to the stack (R4 stack)
4. R4 set PIVR3_R4_SHUTING_DOWN flag, save current SP into PIVR1, write Warm_Boot_Entry address in PIVR0

How R4 transfers to M3

5. R4 set PIBR flags

6. R4 set M3CR register, make LOAD_M3_TCM = 0 and SOFTRESET_M3 = 0 (M3 start to run)

6.1 M3 will modify the flags in PIBR

7. R4 read the mailbox value in PIBR

If mailbox value == 2, PIVR3 = 0x22(PIVR3_R4_DOWN) and clear the mailbox value in PIBR

8. R4 run WFI, enter “close” state

Crash Scenario 1

R0=0xEA195989
R1=0x00DBB578
R2=0x00000000
R3=0xEA195989
R4=0x016075E0
R5=0x00000000
R6=0x00000000
R7=0x00000000
R8=0x00000000
R9=0x00000000
R10=0x00DB23F8
R11=0x00DBB4FC
R12=0x00DBB500
R13=0x0000F260
R14=0x006475CC
R15=0x00656674

DFSR=0x00000000
DFAR=0xEA195989
IFSR=0x00000000
IFAR=0x00000000

backtrace:

0x006475C4 (exception)
0x00648E58
0x0005DF34
0x002C5294
0x002C591C
0x002B8AE0
0x009FF890
0x009F78FC

abort isr

oem/marvell/marvell_6110_mfp_sdk/init/src/startup.S

```
abort_isr:
    STMDB sp,{r0-r15}       // store regs before modifying
    MOV r0,lr               // pass exception's lr as subroutine param
    BL print_debug_info     // call subroutine to output debug info

    LDR r0,abort_isr_str      <=== load "***abort isr***" string

    BL print_string
    SUB r0,sp,#64           // restore regs r0-r14
    LDM r0,{r0-r14}
    B uart_string_exit
    b      abort_isr


abort_isr_str:
    .word   ABT_string

ABT_string:
.string "\n\r***abort isr***\n\r"
```

Occur when program try to access the invalid address

Call Stack

backtrace:

0x006475C4 (exception)
0x00648E58
0x0005DF34
0x002C5294
0x002C591C
0x002B8AE0
0x009FF890
0x009F78FC

It seems call stack has not been destroyed. Why ?
Because the addresses in backtrace are located in text segment.

Get the call down chain

arm-marvell-eabi-nm marvell_6110_mfp_sdk-debug.elf | sort | > symbol.txt

```
00000000 a FP_OFFSET
00000000 T reset
00000013 a ENABLE_CPSR
00000013 a INITIAL_CPSR
0000003f a INT_MASK
00000092 a IRQ_MODE
00000093 a DISABLE_CPSR
00000093 a DISABLE_CPSR
00000093 a SVC_MODE
000000d1 a FIQ_MODE
000000d2 a IRQ_MODE
000000d3 a SVC_MODE
00003ff8 A __stack__
……
```

"T"
"t" The symbol is in the text (code) section.

T --- export symbol

Get the call down chain

Search 00648E58

```
00647ba4 t _float
0064834c t _getnum
006483e0 T vfstrfmt    <=== 00648E58
00649064 t _sprintf
006490a0 T strfmt
00649160 t strfmt_test
```

Get the call down chain

Search 0005DF34

0005deb4 T dbg_printf<=== 0005DF34

0005df8c t _stdout

0005e050 T dbg_sync

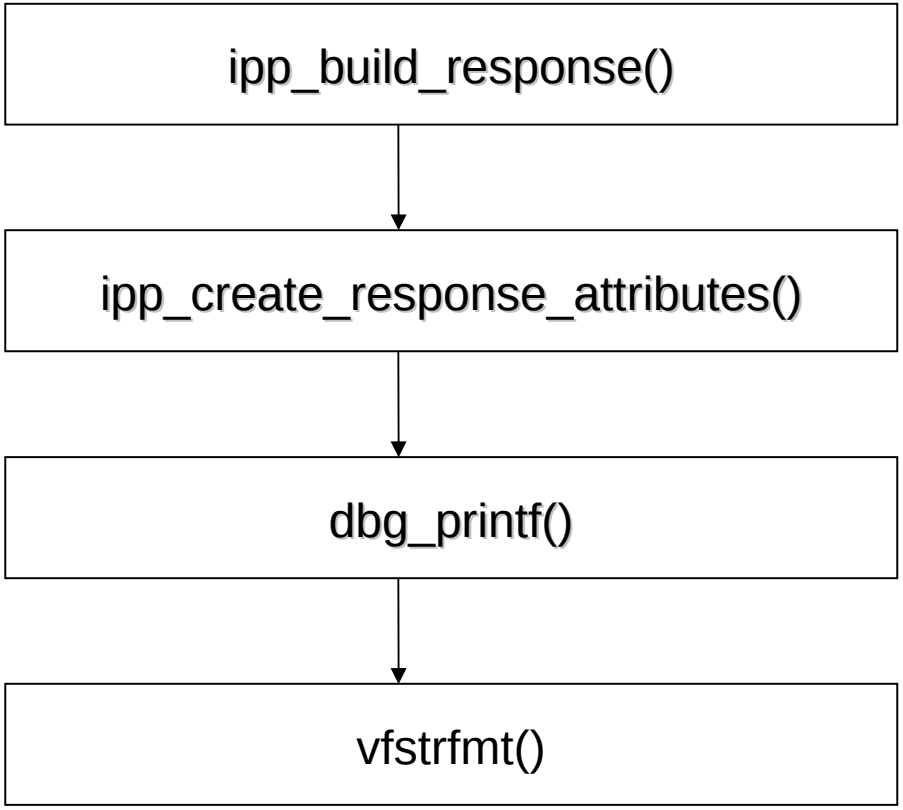
Get the call down chain

Search

0x002C5294
0x002C591C

002c5090 T ipp_create_response_attributes <=== 0x002C5294
002c58b8 T ipp_build_response <=== 0x002C591C
002c5c14 t get_printer_uri
002c5cf8 T printer_uri
002c5df4 T is_ipp_print
002c5e44 T is_ipp_faxout
002c5e94 t _close_faxout_job

Get the call down chain



How to locate the bug

```
2c5234:    e51b102c    ldr    r1, [fp, #-44]
2c5238:    ebffa395    bl     2ae094 <ipp_add_attr_group>
2c523c:    e51b002c    ldr    r0, [fp, #-44]
2c5240:    e3a01001    mov    r1, #1  ; 0x1
2c5244:    ebffa43c    bl     2ae33c <ipp_free_attr_group>
2c5248:    ea000188    b      2c5870 <ipp_create_response_attributes+0x7e0>
2c524c:    e51b0b10    ldr    r0, [fp, #-2832]
2c5250:    eb0002fb    bl     2c5e44 <is_ipp_faxout>      <=== identifier
2c5254:    e1a03000    mov    r3, r0
2c5258:    e3530000    cmp    r3, #0  ; 0x0
2c525c:    0a000019    beq    2c52c8 <ipp_create_response_attributes+0x238>
2c5260:    e59f3624    ldr    r3, [pc, #1572] ; 2c588c <ipp_create_response_attributes+0x7fc>
2c5264:    e59320b4    ldr    r2, [r3, #180]
2c5268:    e3013005    movw   r3, #4101       ; 0x1005
2c526c:    e0023003    and    r3, r2, r3
2c5270:    e3530000    cmp    r3, #0  ; 0x0
2c5274:    0a000007    beq    2c5298 <ipp_create_response_attributes+0x208>
2c5278:    e51b3028    ldr    r3, [fp, #-40]
2c527c:    e5932000    ldr    r2, [r3]
2c5280:    e51b3028    ldr    r3, [fp, #-40]
2c5284:    e5933004    ldr    r3, [r3, #4]
2c5288:    e59f0600    ldr    r0, [pc, #1536] ; 2c5890 <ipp_create_response_attributes+0x800>
2c528c:    e1a01002    mov    r1, r2
2c5290:    e1a02003    mov    r2, r3
2c5294:    ebf66306    bl     5deb4 <dbg_printf> <=== 0x002C5294, the invocation will crash
2c5298:    e51b3b10    ldr    r3, [fp, #-2832]
2c529c:    e593202c    ldr    r2, [r3, #44]
```

How to locate the bug

```
void ipp_create_response_attributes(ipp_request_t *ipp_req)
{
    ipp_cntxt_t *cntxt = ipp_req->ipp_ctxt;
    ASSERT(cntxt);

    ipp_attr_grp_t     *grp;

    ASSERT(ipp_req);
      ......
       case IPP_OPID_GET_JOB_ATTR:
       case IPP_OPID_PRINT_JOB:
       case IPP_OPID_CREATE_JOB:
       case IPP_OPID_SEND_DOCUMENT:
       case IPP_OPID_CLOSE_JOB:
       {
           //
           // In the future, we should move print service job attributes into the hash table, but now, we
           // have to fork to handle print and faxout. (I don't want impact the original code violently)
           //
           //
           if(is_ipp_faxout(ipp_req))              <=== identifier
           {
               DBG_VERBOSE("requested_attributes = (%d, %s)\n", requested_attributes-
      >num_req_attr, requested_attributes->req_attr_str);   <=== make crash
               grp = _faxout_job_attributes(cntxt, ipp_req->job_id, ipp_req->ipp_status,
      requested_attributes);
```

root cause

Search **0x006475C4 (exception)**

```
00647564 <_str>:
  647564:    e1a0c00d    mov    ip, sp
  647568:    e92dd800    push   {fp, ip, lr, pc}
  64756c:    e24cb004    sub    fp, ip, #4       ; 0x4
  647570:    e24dd020    sub    sp, sp, #32      ; 0x20
  647574:    e50b0020    str    r0, [fp, #-32]
  647578:    e50b1024    str    r1, [fp, #-36]
  64757c:    e50b2028    str    r2, [fp, #-40]
  647580:    e51b3028    ldr    r3, [fp, #-40]
  647584:    e3530000    cmp    r3, #0  ; 0x0
  647588:    0a000004    beq    6475a0 <_str+0x3c>
  64758c:    e3a03001    mov    r3, #1  ; 0x1
  647590:    e50b3018    str    r3, [fp, #-24]
  647594:    e3a03000    mov    r3, #0  ; 0x0
  647598:    e50b3010    str    r3, [fp, #-16]
  64759c:    ea000012    b      6475ec <_str+0x88>
  6475a0:    e3a03000    mov    r3, #0  ; 0x0
  6475a4:    e50b3018    str    r3, [fp, #-24]
  6475a8:    e51b3020    ldr    r3, [fp, #-32]

...

  6475b8:    e2833001    add    r3, r3, #1       ; 0x1
  6475bc:    e50b3018    str    r3, [fp, #-24]
  6475c0:    e51b3010    ldr    r3, [fp, #-16]
  6475c4:    e5d33000    ldrb   r3, [r3]<=== this instruction make system crash, [r3] point  to
             invalid address
  6475c8:    e3530000    cmp    r3, #0  ; 0x0
  6475cc:    03a03000    moveq          r3, #0  ; 0x0
```

Crash Scenario 2

backtrace:

0x00F8D7E4 (exception)

0x000000AD

abort isr

The stack has been

destroyed !

Search hints

0x00f8d8a8 unirast_tx_thread

0x00f8d740 unirast_stack

0x00F8D7E4 is near unirast_stack symbol

Some thread(s) has destroyed the thread stack of urf parser, make urf parser crash

Understanding Unit Test Framework

```
static int cdma_test(void)
{
        uint32_t start1, start2, end1, end2, test_size = DMA_TEST_SIZE, i;
        bool flag = FALSE;
        cdma_handle_t *handle = NULL;
        CDMA_CONFIG MyCfg;
        uint8_t *destbuf, *srcbuf;
        int result = 0;
        while(1)
        {
        destbuf = (uint8_t *)MEM_MALLOC_ALIGN((sizeof(uint8_t) * DMA_TEST_SIZE),
        cpu_get_dcache_line_size());
        if(destbuf)
                        break;
        tx_thread_sleep(20);
        }
        ……
}
UNIT_TEST("cdma", cdma_test)’
```

CMD==> test run cdma cdma_test

Understanding Unit Test Framework

#define UNIT_TEST(g, fn) static struct utest_ent utest_##fn __attribute__((__section__(".utest"))) __attribute__((used)) = {fn, #fn, g}

For example:

UNIT_TEST("cdma", cdma_test);

➔ **static struct utest_ent utest_cdma __attribute__((__section__(".utest"))) __attribute__((used)) = {cdma_test, "cdma_test", "cdma"};**

```
struct utest_ent{
    utest_t fn;
    char *name;
    char *group;
};
```

"UNIT_TEST" macro create a utest_ent entry, and the entry will be put into "**.utest**" section.

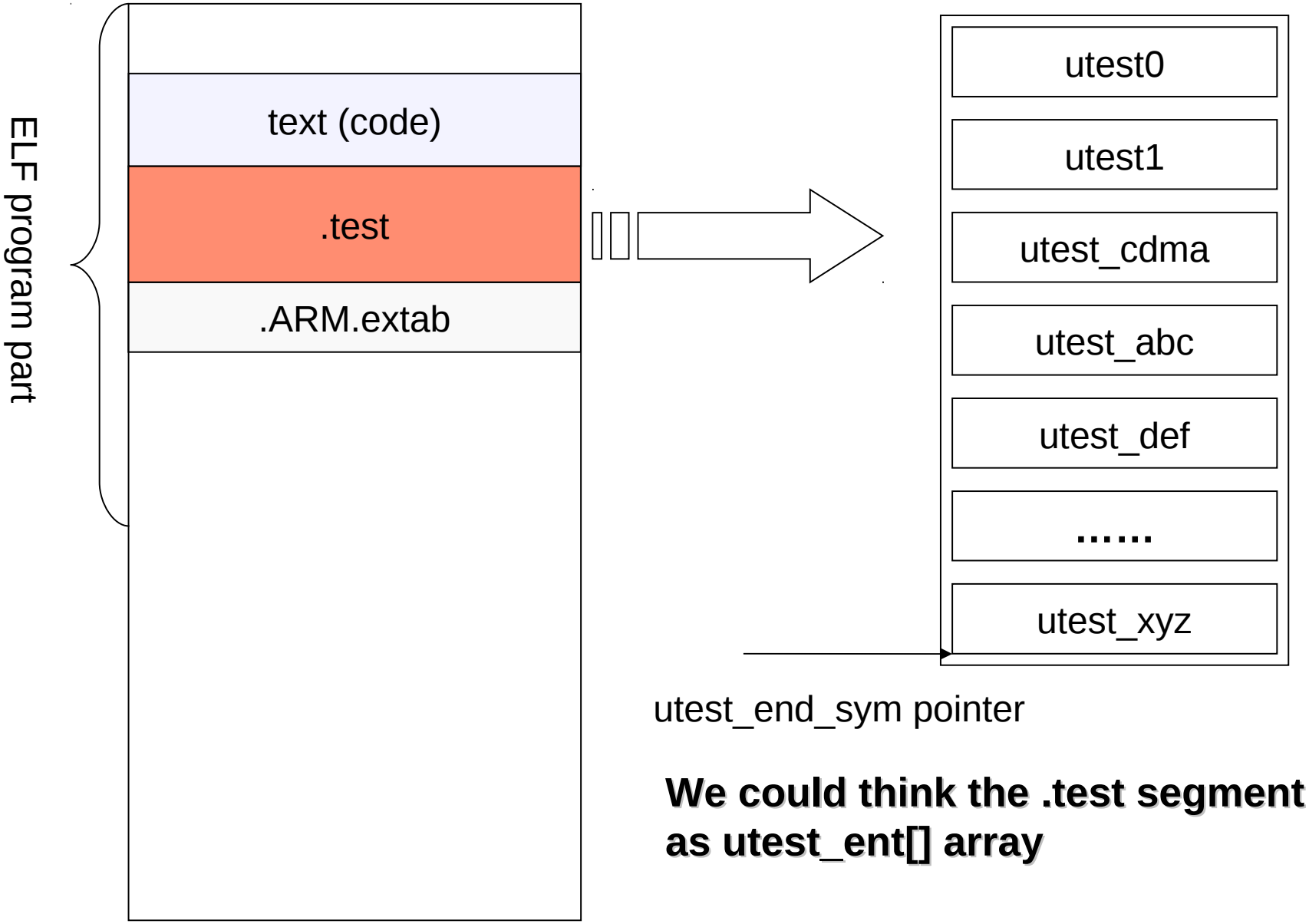
Understanding Unit Test Framework

In the linker script --- memory_map.ld

```
/* Add below for unit test code */
.test : {
        *(.utest0)
        *(.utest1)
        *(.utest)
        utest_end_sym = .;
 } > ram
```

All utest_ent entries created by UNIT_TEST macro is in .test segment. The .utest0 is the header of the segment, and the utest_end_sym is the tail of the segment

Understanding Unit Test Framework



We could think the .test segment as utest_ent[] array

Reference

Linker script introduction: <Linker Editor Chinese Edition>,
http://wzhou1997.0catch.com/translation/Linker-Editor.pdf

ELF format analyse document:

http://wzhou1997.0catch.com/Linux/Kernel/obj-file-relocation.pdf

http://wzhou1997.0catch.com/Linux/Kernel/symbol-analyse.pdf

ELF loader introduction

http://pan.baidu.com/s/1jGEFZzo

Q & A